بسم الله الرحمن الرحیم

پروژه تشخیص سرطان سینه

استاد:مهندس تورانی

اعضای گروه:

سیده مریم میرصالحی

مهسا غلامی

فاطمه بابایی

سپیده یوسفی

گزارش فعالیت ها:

ابتدا با زبان پایتون آشنا شدیم.

و در مرحله بعد به جستجو در اینترنت و جمع آوری اطلاعات پرداختیم.

یادگیری عمیق یا همان **یادگیری ژرف** یا **Deep Learning** یکی از مباحث در هوش مصنوعی و یادگیری ماشین است.

در یک تعریف کلی، یادگیری عمیق، همان یادگیری ماشین است، به طوری که در سطوح مختلف نمایش یا انتزاع (abstraction) یادگیری را برای ماشین انجام میدهد. با این کار، ماشین درک بهتری از واقعیت وجودی داده ها پیدا کرده و میتواند الگوهای مختلف را شناسایی کند.

برای شناسایی یادگیری عمیق، ابتدا نیاز به دانستن شبکه های عصبی دارید. بر اساس تعریف مشهور، یادگیری عمیق در واقع همان یادگیری به وسیله شبکه های عصبی ای هستند که دارای لایه های ی پنهانی (Hidden Layers) زیادی می باشند. هر چقدر در لایه های یک شبکه عصبی عمیق جلو تر می رویم، به مدلهای پیچیده تر (و کامل تری) میرسیم.

یادگیری عمیق (Deep Learning) در حوزه های مختلفی مانند، دسته بندی تصاویر (تشخیص تصاویر)، دسته بندی متون، تشخیص صدا و... کاربردهای فراوانی دارد.

که ما این پروژه را با استفاده از Deep Learning انجام می دهیم.

Dataset: برای دانلود دیتاست وارد سایت https://www.kaggle.com/ می شویم و یک اکانت می سازیم. و فایل json دیتاست را دانلود می کنیم.

▾ downloading data

```
[ ]  !pip install -q kaggle
     !mkdir ~/.kaggle
     !cp kaggle.json ~/.kaggle/
     !chmod 600 ~/.kaggle/kaggle.json
```

```
[ ]  !kaggle datasets download -d paultimothymooney/breast-histopathology-images
```

```
[ ]  !mkdir dataset
     !unzip breast-histopathology-images.zip -d dataset
```

کتابخانه های مورد نیاز:

```
[ ]  import os
     import copy
     from glob import glob
     import fnmatch
     import time

     import numpy as np
     import pandas as pd
     import cv2
     import random
     import seaborn as sns
     import matplotlib.pyplot as plt
     %matplotlib inline

     from sklearn.model_selection import train_test_split
     from sklearn.metrics import classification_report
     from sklearn.metrics import confusion_matrix

     import torch
     import torch.nn as nn
     import torch.nn.functional as F
     import torchvision
     import torch.optim as optim
     import torchvision.transforms as transforms
     from torch.utils.data import TensorDataset, DataLoader, Dataset
```

:import time،import fnmatch،from glob import glob،import copy،import os

این کتابخانه ها امکان استفاده از برخی از قابلیت های سیستم عامل،برای کار با stringها و آدرس فایل ها را فراهم می کنند.

numpy: یکی از اصلی ترین پکیج های محاسبات علمی و عددی در پایتون است که امکان ایجاد آرایه،ماتریس ها،مجموعه ها و.... را به ما می دهد.

pandas: یکی از ابزارهای رایج Data scientists برای انجام تجزیه و تحلیل داده ها است.یک کتابخانه پایتون است که ساختارهای داده سریع، انعطاف‌پذیر و رسای طراحی شده برای کار با داده‌های "رابطه‌ای" (Relational) یا برچسب دار (Labeled)را با سادگی و بینش فراهم می‌کند. هدف از این کتابخانه آن است که یک بلوک سازنده سطح بالای اساسی برای انجام تحلیل‌های "جهان واقعی" (Real World) و کاربردی در پایتون فراهم کند. و این کتابخانه برای نگه داشتن دیکشنری و فایل هایی که به صورت سطر و ستون هستند مناسب می باشد.

cv2: یکی از کتابخانه های اصلی پردازش تصویر می باشد.

random: random برای ایجاد انتخاب های تصادفی در پایتون استفاده می شود. انتخاب ها میتوانند یک عدد یا string و یا یک لیست باشند.

seaborn: یک کتابخانه بسیار مفید مصورسازی داده در پایتون محسوب می گردد.

matplotlib.pyplot: یک کتابخانه گرافیکی برای ترسیم نمودارهای دو بعدی است.

torch: همان pytorch است کتابخانه اصلی Deep Learning است.

پای‌تورچ (PyTorch) یک بسته پایتون است که دو ویژگی سطح بالا که در زیر بیان شده‌اند را فراهم می‌کند.

- محاسبات تانسور (مانند NumPy) با شتاب‌دهی قدرتمند GPU
- شبکه‌های عصبی عمیق ساخته شده روی یک سیستم مبتنی بر نوارautograd

کاربر می‌تواند از بسته‌های پایتون محبوب خود مانند SciPy ، NumPy جهت توسعه PyTorch در هنگام نیاز استفاده مجدد کند.

torch.nn: nn ماژول های مربوط به خود شبکه های عصبی را دارد.

torchvision.transforms: یکی از کارهای آن crop کردن عکس می باشد.

۵۵۵۰۴۸ تصویر داریم فرمت همه ی تصاویر png می باشد. این خط از کد می آید id و ۰ و ۱ های آن ها را می خواند.

نکته:

صفر (۰):سرطان وجود ندارد (عدم وجود)

صفر (۱):سرطان وجود دارد (وجود)

Recursive تا انتهای پوشه می رود(هر چی پوشه دارد)

preparing data

```
[ ] image_paths = np.array(glob('dataset/**/*.png', recursive=True))
    len(image_paths)
```

```
555048
```

چون تصاویر خیلی زیاد هستند و train آن ها خیلی طول می کشد با استفاده از الگوریتم random ۴/۵(چهار پنجم) عکس ها را کنار گذاشتیم،بنابراین ۱۱۱۰۱۰ تصویر داریم.(با پاک کردن ۴/۵ در کد کل عکس ها train می شوند.)

```
[ ]  # dropping 4/5 of the dataset as it is huge!
     drop_indices = random.sample(set(np.arange(0, len(image_paths))), len(image_paths)*4//5)
     image_paths = np.delete(image_paths, drop_indices)
     len(image_paths)
```

```
111010
```

هر تصویر یک pattern دارد و ته آن تصویر یا کلاس صفر است یا یک بر اساس همین pattern ما label خود را می سازیم label ما می شود یا صفر یا یک.یک لیستی است که به تعداد تصویر های ورودی صفر یا یک است.در مرحله بعد reshape می کنیم چون همه چیز باید ستونی باشد و می خواهیم ماتریس operation انجام دهیم و عملا y یک لیستی است که داده ها در آن به صورت سطری قرار دارند که این به درد ما نمی خورد و ما می خواهیم آن را ستونی کنیم.بنابراین تعداد آن ها ۷۹۶۷۱برای تصویرهایی که سرطان ندارند و ۳۱۳۳۹برای تصویرهایی که سرطان دارند و y.shape ستونی است که ۱۱۱۰۱۰ سطر دارد.

```
[ ]  pattern_zero = '*class0.png'
     pattern_one = '*class1.png'
     class_zero = fnmatch.filter(image_paths, pattern_zero)
     class_one = fnmatch.filter(image_paths, pattern_one)

     y = []
     for img in image_paths:
         if img.__contains__('class0'):
             y.append(0)
         elif img.__contains__('class1'):
             y.append(1)
     y = np.array(y).reshape(-1, 1)

     print(len(class_zero))
     print(len(class_one))
     print(y.shape)
```

```
79386
31624
(111010, 1)
```

یک DataFrame تعریف می کنیم که دو تا ستون دارد image و label.

Image ها آدرس های تصاویر هستند و label ها همان y هستند.ما ۱۱۱۰۱۰ تصویر داریم که می خواهیم تعدادی از آن را برای train کردن استفاده کنیم و تعدادی را هم برای test کردن،تابعی داریم به اسم train_test_split که دیتاها را به آن می دهیم اگر

این کار را به صورت random انجام دهیم ممکن است نسبت ها به هم بخورد از stratify استفاده کردیم تا نسبتی که در train هست در test هم همان باشد.

```
[ ]  images_df = pd.DataFrame()
     images_df["images"] = image_paths
     images_df["labels"] = y

     train, test = train_test_split(images_df, stratify=images_df.labels, test_size=0.2)
     len(train), len(test)
```

```
(88808, 22202)
```

یک کلاس تعریف کردیم به نام BreastCancerDataset(Dataset) که دیتا را به آن می دهیم. متد len طول کل تصویر ها را می دهد و getitem برای زمانی هست که یک تصویر بخواهیم.self.data[index] آدرس و label تصویر را می گیرد و با cv2،تصویر را می خواند.و یک آبجکتی به اسم transform.compose می سازیم که داخل آن می تواند عکس را resize کند.

custom dataset

```
[ ]  class BreastCancerDataset(Dataset):
         def __init__(self, data, transforms=None):
             super().__init__()
             self.data = data
             self.transforms = transforms

         def __len__(self):
             return len(self.data)

         def __getitem__(self, index):
             img_path, label = self.data[index]
             image = cv2.imread(img_path)

             if self.transforms is not None:
                 image = self.transforms(image)

             return image, label
```

کلاس ها را تعریف می کنیم(۲ کلاس) که خروجی های آن ها به صورت احتمال می باشد.

```
[ ]  num_epochs = 30
     num_classes = 2
     batch_size = 512

     device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
```

transforms.resize اندازه تصاویر را ۶۴ در ۶۴ می کند.

transforms.RandomHorizontalFlip تصویر را در جهت افقی معکوس می کند.

transforms.RandomVerticalFlip تصویر را در جهت عمودی معکوس می کند.

transforms.RandomRotation تصویر را بین ۲۰- تا ۲۰ درجه rotate می کند.

و در نهایت برای پردازش برداری و سرعت بالا و چون می خواهیم به gpu بفرستیم از Tensorاستفاده می کنی،از اینجا به بعد دیگه آرایه نیستند Tensor هستند و بعد Normalizeمی کنیم.(وقتی تصاویر را می دهیم رنج هر کدام بین ۰ تا ۲۵۵ است سه تا حالت داریم RGB)

در حالت Test فقط Resize و Normalize می کنیم.

DataLoader کار Parallel انجام دادن چند تا تصویر و Load کردن و Transport کردن آن ها را انجام می دهد.

batch_size: یعنی هر بار وارد کلاس می شود چند تا عکس load شود که کارهایی که گفته شود روی آن ها صورت بگیرد که ما آن را روی ۵۱۲ گذاشتیم.

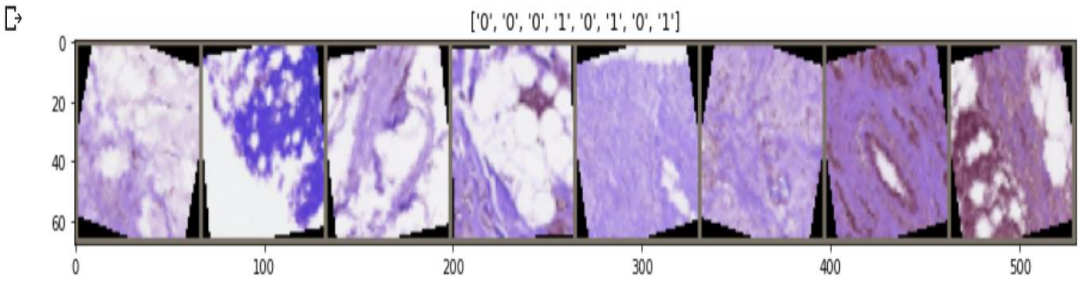
حتما باید shuffle انجام دهیم چون shuffle ترتیب را عوض می کند اگر ترتیب را عوض نکنیم ممکن است دیتای logic دار تولید شود.

```
[ ]  trans_train = transforms.Compose([transforms.ToPILImage(),
                                       transforms.Resize((64, 64)),
                                       transforms.RandomHorizontalFlip(),
                                       transforms.RandomVerticalFlip(),
                                       transforms.RandomRotation(20),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

     trans_test = transforms.Compose([transforms.ToPILImage(),
                                       transforms.Resize((64, 64)),
                                       transforms.ToTensor(),
                                       transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

     train_dataset = BreastCancerDataset(data=train.values, transforms=trans_train)
     test_dataset = BreastCancerDataset(data=test.values,transforms=trans_test)

     train_loader = DataLoader(dataset = train_dataset, batch_size=batch_size, shuffle=True, num_workers=0)
     test_loader = DataLoader(dataset = test_dataset, batch_size=batch_size//2, shuffle=False, num_workers=0)
     dataloaders = {'train': train_loader, 'test': test_loader}
```

این قسمت تصویر را نشان می دهد و وجود سرطان و عدم وجود سرطان را مشخص می کند.

Tensor، numpy نیست torch است چون torch است تبدیلش می کنیم به numpy که محاسبات را انجام دهد و هر عکس با لیبل خودش match می شود،و وجود سرطان را با لیبل یک و عدم وجود را با لیبل صفر نشان می دهد.

```
[ ]  class_names= ['0', '1']
     def imshow(inp, title=None):
         inp = inp.numpy().transpose((1, 2, 0))
         mean = np.array([0.485, 0.456, 0.406])
         std = np.array([0.229, 0.224, 0.225])
         inp = std * inp + mean
         inp = np.clip(inp, 0, 1)
         plt.imshow(inp)
         if title is not None:
             plt.title(title)
         plt.pause(0.001)  # pause a bit so that plots are updated


     # Get a batch of training data
     inputs, classes = next(iter(dataloaders['train']))

     # Make a grid from batch
     out = torchvision.utils.make_grid(inputs[:8,:,:,:])

     plt.figure(figsize=(15, 10))
     imshow(out, title=[class_names[x] for x in classes[:8]])
```



resnet101 مشهورترین مدل برای transfer learning است.

resnet34 ۳۴ تا لایه دارد.

این کد مدلی که وزن هایی که از قبل آموزش دیده شدن برای کار image net را به ما می دهد.

loading pretrained resnet 101

```
[ ]  # fine tune resnet 101 model trained on image net
     model = torchvision.models.resnet34(pretrained=True)
```

CrossEntropyLoss در multiclassification استفاده می شود که اینجا دوتاست.

برای Adam،Optimizer استفاده می کنیم که مشتق Lossfunction را به دست می آورد و با استفاده از گرادیان آن را هی کم می کند.

ما به اندازه گرادیان جا به جا نمی شویم.هر ۷بار که epoch حساب می شود، Learning Rate را با یک gamma که ضریبی از خود LearningRate هست کم می کنیم.

نکته:epoch یعنی چند بار کل دیتا را از اول تا آخر به شبکه بدهیم تا error های آن را حساب کند.

epoch 1 یعنی یکبار کل دیتا را به شبکه بدهیم.

epoch 2 یعنی یکبار که دادیم برگشت و همه ی دیتاهای شبکه را دید دوباره همان را بدهیم با یک ترتیب به هم ریخته دیگر.

```
[ ]  fc_in_features = model.fc.in_features
     model.fc = nn.Linear(fc_in_features, num_classes)
     model = model.to(device)

     criterion = nn.CrossEntropyLoss()
     optimizer = optim.Adam(model.parameters())
     exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=7, gamma=0.1)
```

با استفاده از copy.deepcopy(model.state_dict()) وزن ها ذخیره می کنیم.

دو تا فاز داریم:۱-فاز train ۲-فاز test

در فاز train مدل را می گذاریم در حالت train و در فاز test مدل را می گذاریم در حالت eval.

در حالت eval وزن ها را آپدیت نمی کنیم چون نمی خواهیم مدلمون testcase ها را یاد بگیرد.

```
[ ]  def train_model(model, criterion, optimizer, scheduler, num_epochs=30):
         best_model_wts = copy.deepcopy(model.state_dict())
         best_acc = 0.0
         since = time.time()

         for epoch in range(num_epochs):
             print('Epoch {}/{}'.format(epoch, num_epochs - 1))
             print('-' * 10)

             # Each epoch has a training and validation phase
             for phase in ['train', 'test']:
                 if phase == 'train':
                     model.train()  # Set model to training mode
                 else:
                     model.eval()   # Set model to evaluate mode

                 running_loss = 0.0
                 running_corrects = 0
```

dataloader کارهای خواندن دیتا را انجام می دهد.

حلقه ی for،۵۱۲ تا ۵۱۲ تا دیتاها را به ما می دهد.(input ۵۱۲ تا image است و label ۵۱۲ تا label) که to. اینا ها را می فرستد به gpu

```
# Iterate over data.
for inputs, labels in dataloaders[phase]:
    inputs = inputs.to(device)
    labels = labels.to(device)
```

چون وزن آپدیت شده است دیگر error های قبلی به درد ما نمی خورد ما باید ببینیم errorجدید متناسب با وزن جدید چیست هر error اسمش گرادیان است با هر بار iteration یکبار گرادیان را با optimizer.zero_grad() صفر می کنیم.

```
# zero the parameter gradients
optimizer.zero_grad()
```

torch.set_grad_enabled یعنی برو گرادیان را حساب کن در model.eval گرادیان را حساب نمی کنیم.به مدلمون input ها را می دهیم که تا آخر می رود و prediction ما به دست می آید.در max منظور از یک این است که سطری max بگیر اگر ۱ نذاریم ستونی max می گیرد.در loss نیز criterion را حساب می کنیم و میریزیم.

```
# forward
# track history if only in train
with torch.set_grad_enabled(phase == 'train'):
    outputs = model(inputs)
    _, preds = torch.max(outputs, 1)
    loss = criterion(outputs, labels)
```

حالا backward را انجام می دهیم در حالت train، backward می کنیم در حالت test، backward نمی کنیم.در حالت train وقتی می خواهیم backward کنیم با استفاده از loss.backward() مشتقات را حساب می کنیم (همان گرادیان) و با optimizer.step() وزن ها را آپدیت می کنیم.

schedular.step():هر ۷ بار یکبار که fit forward می کنیم یکبار learning rate را کم می کنیم.

```
        # backward + optimize only if in training phase
        if phase == 'train':
            loss.backward()
            optimizer.step()

    # statistics
    running_loss += loss.item() * inputs.size(0)
    running_corrects += torch.sum(preds == labels.data)
if phase == 'train':
    scheduler.step()
```

این قسمت از کد loss را حساب می کند.کل loss را تقسیم بر دیتاست می کنیم که میانگین loss به دست می آید.

```
epoch_loss = running_loss / dataset_sizes[phase]
epoch_acc = running_corrects.double() / dataset_sizes[phase]

print('{} Loss: {:.4f} Acc: {:.4f}'.format(
    phase, epoch_loss, epoch_acc))
```

اگر در فاز test بودیم و epoch_acc > best_acc بود مدل جدید وزن را save می کند و بعد محاسبات time را انجام می دهد.و بهترین مدلی که در کل بازه یاد گرفته است را نگه می دارد.

```
            # deep copy the model
            if phase == 'test' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.state_dict())

        print()

    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # load best model weights
    model.load_state_dict(best_model_wts)
    return model
```

در testloss اول ۸۷ بوده با یک epoch به ۸۸ می رسد، و در ۱۷ epoch در یک local minima گیر کرده است.

```
[ ]  Epoch 0/29
     ----------
     train Loss: 0.3462 Acc: 0.8545
     test Loss: 0.2960 Acc: 0.8767

     Epoch 1/29
     ----------
     train Loss: 0.2875 Acc: 0.8783
     test Loss: 0.2759 Acc: 0.8835

     Epoch 2/29
     ----------
     train Loss: 0.2782 Acc: 0.8824
     test Loss: 0.3416 Acc: 0.8586

     Epoch 3/29
     ----------
     train Loss: 0.2663 Acc: 0.8874
     test Loss: 0.2654 Acc: 0.8882

     Epoch 4/29
     ----------
     train Loss: 0.2634 Acc: 0.8888
     test Loss: 0.2724 Acc: 0.8848
```

```
[ ]  Epoch 5/29
     ----------
     train Loss: 0.2552 Acc: 0.8932
     test Loss: 0.2760 Acc: 0.8854

     Epoch 6/29
     ----------
     train Loss: 0.2499 Acc: 0.8950
     test Loss: 0.2585 Acc: 0.8903

     Epoch 7/29
     ----------
     train Loss: 0.2299 Acc: 0.9032
     test Loss: 0.2251 Acc: 0.9058

     Epoch 8/29
     ----------
     train Loss: 0.2237 Acc: 0.9062
     test Loss: 0.2228 Acc: 0.9058

     Epoch 9/29
     ----------
     train Loss: 0.2202 Acc: 0.9077
     test Loss: 0.2239 Acc: 0.9080
```

```
[ ]
     Epoch 10/29
     ----------
     train Loss: 0.2166 Acc: 0.9093
     test Loss: 0.2207 Acc: 0.9078

     Epoch 11/29
     ----------
     train Loss: 0.2137 Acc: 0.9106
     test Loss: 0.2189 Acc: 0.9078

     Epoch 12/29
     ----------
     train Loss: 0.2118 Acc: 0.9103
     test Loss: 0.2169 Acc: 0.9088

     Epoch 13/29
     ----------
     train Loss: 0.2081 Acc: 0.9122
     test Loss: 0.2191 Acc: 0.9105

     Epoch 14/29
     ----------
     train Loss: 0.2027 Acc: 0.9157
     test Loss: 0.2144 Acc: 0.9111
```

```
Epoch 15/29
----------
train Loss: 0.2009 Acc: 0.9167
test Loss: 0.2144 Acc: 0.9110

Epoch 16/29
----------
train Loss: 0.1996 Acc: 0.9168
test Loss: 0.2140 Acc: 0.9105

Epoch 17/29
----------
train Loss: 0.1987 Acc: 0.9174
test Loss: 0.2146 Acc: 0.9118

Epoch 18/29
----------
train Loss: 0.1988 Acc: 0.9164
test Loss: 0.2149 Acc: 0.9107

Epoch 19/29
----------
train Loss: 0.1984 Acc: 0.9174
test Loss: 0.2136 Acc: 0.9111
```

```
Epoch 20/29
----------
train Loss: 0.1976 Acc: 0.9171
test Loss: 0.2141 Acc: 0.9112

Epoch 21/29
----------
train Loss: 0.1954 Acc: 0.9191
test Loss: 0.2139 Acc: 0.9117

Epoch 22/29
----------
train Loss: 0.1967 Acc: 0.9177
test Loss: 0.2143 Acc: 0.9112

Epoch 23/29
----------
train Loss: 0.1961 Acc: 0.9179
test Loss: 0.2142 Acc: 0.9115

Epoch 24/29
----------
train Loss: 0.1958 Acc: 0.9191
test Loss: 0.2142 Acc: 0.9119
```

```
----------
train Loss: 0.1965 Acc: 0.9177
test Loss: 0.2142 Acc: 0.9109

Epoch 26/29
----------
train Loss: 0.1960 Acc: 0.9178
test Loss: 0.2143 Acc: 0.9110

Epoch 27/29
----------
train Loss: 0.1955 Acc: 0.9177
test Loss: 0.2141 Acc: 0.9110

Epoch 28/29
----------
train Loss: 0.1961 Acc: 0.9172
test Loss: 0.2144 Acc: 0.9113

Epoch 29/29
----------
train Loss: 0.1962 Acc: 0.9180
test Loss: 0.2146 Acc: 0.9114

Training complete in 51m 35s
Best val Acc: 0.911900
```

Visualize_model همان کار test را انجام می دهد فقط عکس ها را نشان می دهد.

گرادیان ها را حساب نمی کنیم،نمی خواهیم چیزی را آپدیت کنیم فقط می خواهیم test کنیم.

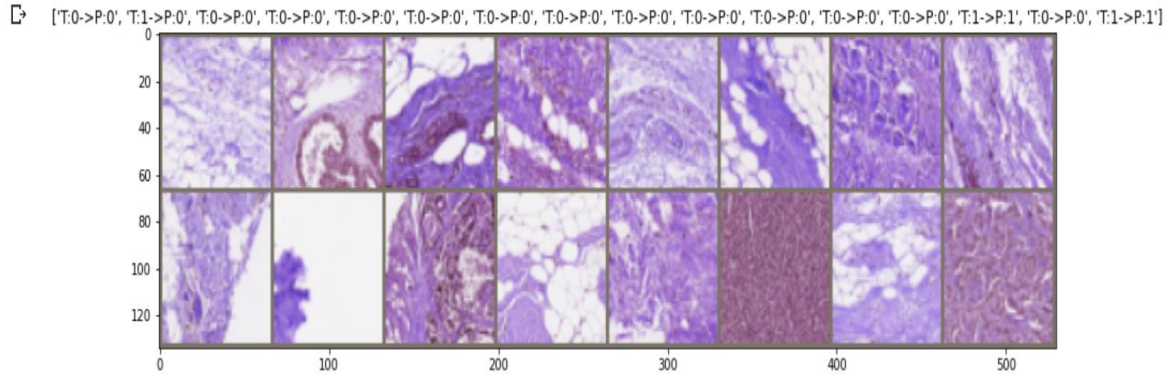
```
lize_model(model, num_images=16):
raining = model.training
.eval()

torch.no_grad():
or i, (inputs, labels) in enumerate(dataloaders['test']):
   inputs = inputs.to(device)
   labels = labels.to(device)

   outputs = model(inputs)
   _, preds = torch.max(outputs, 1)

   out = torchvision.utils.make_grid(inputs[:num_images, :, :, :].cpu())
   plt.figure(figsize=(15, 10))
   imshow(out, title=['T:'+str(class_names[lbl])+'->P:'+str(class_names[p]) for lbl, p in zip(labels[:num_images], preds[:num_images])])
   break
odel.train(mode=was_training)
```

پیش بینی های مدل ما

T یعنی true table

P یعنی prediction

```
[ ] visualize_model(model)
```



برای مثال برای تصویر اول true table آن صفر می باشد یعنی سرطانی نیست و prediction آن نیز صفر آمده است بنابراین درست تشخیص داده است.

پایان